Visualizing I/O Performance During the BGL Deployment[*†]

Andrew Uselton, Brian Behlendorf

March 13, 2007

Abstract

Among the many challenges in getting BlueGene/L into production was the initial under-performance of the Lustre-based parallel file system. This report focuses on the role of visualization tools that assisted in understanding and improving performance. The authors were able to bring about a two-fold increase in the delivered I/O performance.[1]

1 Introduction

Once the BlueGene/L[4, 1] (BGL) hardware was fully deployed at Lawrence Livermore National Lab (LLNL), along with its I/O subsystem, it became apparent that the peak sustainable I/O rate from BGL to its Lustre[3, 8, 5]-based mass storage file system was well below half the rate initially expected[6]. This observation was met with a certain amount of alarm by all involved. This paper presents three examples where a graphical representation of the observed data rates was instrumental in understanding and improving performance.

The initial roll-out of BGL has been presented elsewhere[6], and its construction will not be detailed here except in the most general terms. As depicted in Figure 1, BGL presents itself for I/O as a 1024 node cluster, with each node

[*]UCRL-CONF-226766

[†]This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

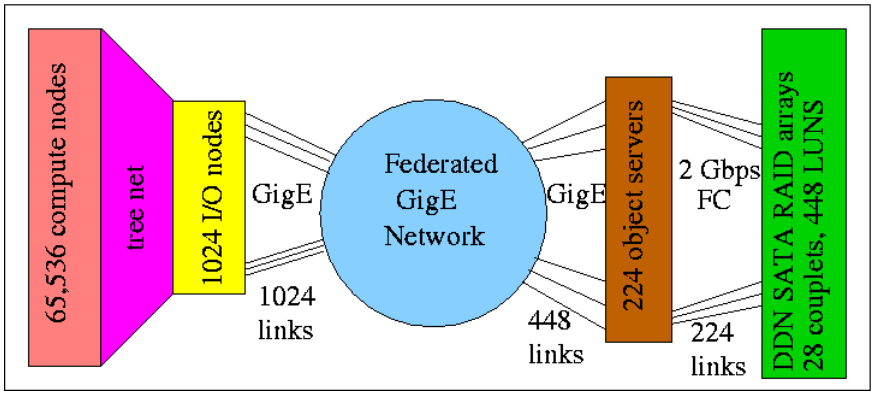


Figure 1: BGL and its storage cluster

attached via GigE links to a federated network of switches. A 224 node storage cluster serves the Lustre parallel file system, and connects to the federated switch network via 448 GigE links - two per server. Each server provides storage from each of two 2 TB LUNs provided by a DDN RAID enclosure (28 DDNs total), and is connected to the DDN via a 2 Gb/s fiberchannel link. The bandwidth through the network was initially estimated to be about 45 GB/s. Under a random workload a DDN could write a maximum of about 1.2 GB/s and read a little over 800 MB/s[2]. Thus the aggregate bandwidth of the disk back-end was about 35 GB/s for writes and 24 GB/s for reads. The DDNs are the primary constraint on the over-all expected performance.

When initial file system testing showed no more than 12 GB/s for writes and even less for reads there was much discussion about where the performance was being lost.

The rest of this paper presents three incremental improvements to the I/O performance of BGL. In each case visualizing the performance test results helped in both understanding and addressing the under-performance. Section 2 discusses measuring the performance of the GigE federated network by presenting results in a node-by-node graph. Section 3 identifies the effect of "stragglers" on file system performance and the role of *lmt* in identifying the problem. Also introduced is a new technique, which we call a "system call profile," whereby the timing of every system call is captured and graphed. Section 4 presents these system call profiles in a composite graph together with the measured, aggregate data rate and server-side performance measurements provided by the *lmt* tool. This composite view was useful in identifying a tuning issue that allowed the read rate to achieve a near-optimum value. Section 6 concludes with a few remarks about ideas for additional work.

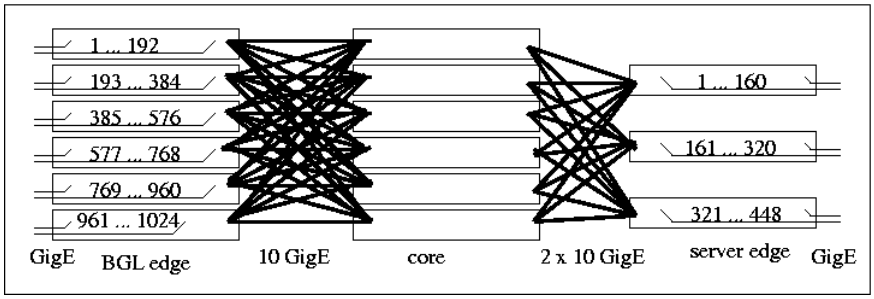


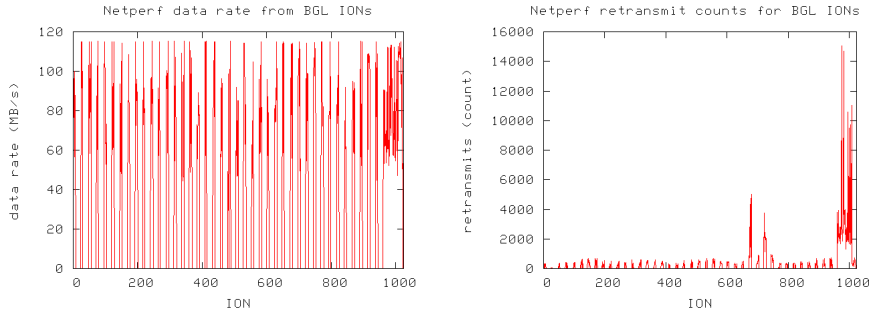
Figure 2: The federated GigE network

2 Network Performance

The theoretical bandwidth of the network was estimated as follows: Each of the six switches directly connected to BGL I/O nodes in Figure 2 had a 10 GigE link to each of the six core switches. Thus, each switch had a theoretical bandwidth of 7.5 GB/s. Similarly, there were two 10 GigE links from each core switch to each of the three server-side switches. The Lustre team used a parallel arrangement of 1024 individual *netperf* tests to document the federated network's aggregate bandwidth. Initial testing showed about 30 GB/s, which meant the network was not the bottleneck for the I/O path. Nevertheless, several experiments were conducted to try to identify why the network was performing at about 2/3 its theoretical capacity.

Testing each switch, one switch at a time, showed a consistent maximum of approximately 6 GB/s per switch. The first five switches each had 192 GigE links to BGL I/O nodes and the sixth switch had 64 GigE links. Thus the five fully populated switches were performing at closer to 80% of theoretical maximum, which is a more tolerable value. The sixth switch was adding little to the total bandwidth.

The Netperf benchmark ran such that every BGL I/O node targeted a server and all servers were more or less evenly targeted. The network was oversubscribed in two ways. First, any one *line card* of 24 GigE ports in a switch could saturate its backplane connection with 20 of the links active. Second, with 8 such line cards in a switch, the six 10 GigE links to the core could be saturated with fewer than 10 links active per line card. Subsequent tests attempted to present a balanced load to the network. Figure 3 shows one such test. Each switch, and each line card in the switch, gets a load that just matches its expected maximum capacity, thus many I/O nodes are idle in the figure. Figure 3(a) shows the data rates recorded for each I/O node, and Figure 3(b) shows the observed retransmit rate. Switch six is a special case, and all 64 of its links are active in the pictured test. Note the higher retransmit rate for the sixth switch (nodes 961 through 1024) where the line cards are still oversubscribed. Before being put into production the GigE links from BGL I/O nodes were moved such that the loading on the switches was more evenly distributed.



(a) data rate

(b) retransmits

Figure 3: Performance of the federated GigE network

Additional experiments attempted to improve network performance by investigating the effect of routing policy, channel balancing, subnet mapping, and routing fairness. The end result was a modest improvement in the network bandwidth and a better understanding of the issues. One final networking issue, TCP/IP retransmit/back-off settings, also proved crucial to improving I/O performance and is discussed in the next section.

An important lesson learned over the course of these experiments was that it is helpful to have a way of visualizing the behavior of the network during the test. This lesson was reinforced in the Lustre team's efforts to understand and address the file system's under-performance, which is the topic of the next two sections.

3 Stragglers

The Lustre parallel file system has 448 LUNs, with each LUN appearing to one of the 224 servers as a 2 TB SCSI device. Early testing used the *IOR*[2] benchmark, an MPI application for measuring parallel file system performance. It ran across all BGL compute nodes (65536 tasks) writing to and reading from Lustre. *IOR* times the I/O from the beginning of the first operation, coordinated across all tasks by a barrier, to the end of the last operation in any task. The aggregate amount of data moved divided by this interval is the data rate reported.

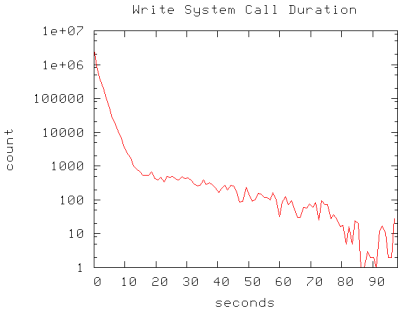
The server monitoring utility *lmt*[3] produces a real-time display of health and performance statistics for the Lustre servers. Figure 4(a) graphs the *lmt* readings from a typical test. First the write rate is plotted for a little over a hundred seconds of testing and then the read test starts and runs for another couple of hundred seconds. Note that the write rate falls off throughout the test.

[2] ftp://ftp.llnl.gov/pub/siop/ior

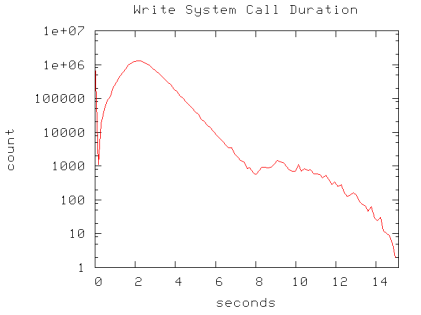
[3] http://sourceforge.net/projects/lmt/

(a) Stragglers lower the aggregate data rate

(b) Stonewalling measures peak data rate

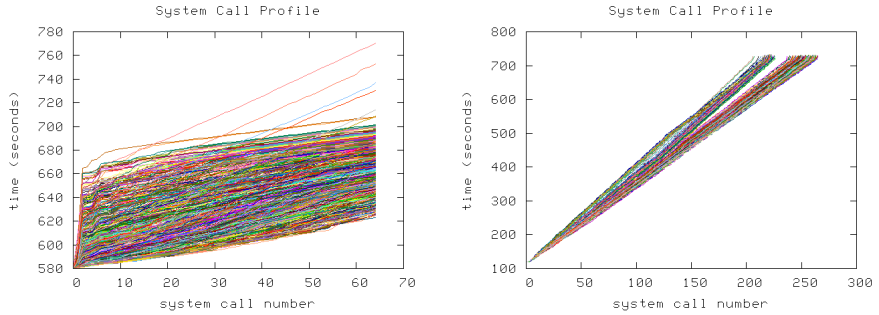


(c) Some system calls take much longer than the average



(d) Both stonewalling and aggressive TCP back-off reduce the effect

Figure 4: Stragglers show up with *lmt* and system call profiles



(a) long back-offs

(b) uneven allocation

Figure 5: Two causes of stragglers

Some tasks lingered performing I/O long after the bulk of the test was complete. The Lustre team sought both to isolate this so-called "stragglers effect" and to explain it. Furthermore, the tendency to have stragglers made test results vary widely between otherwise identical runs.

The test shown in Figure 4(b), which was run about a week later, is rather better behaved, though it still has the same dip early in the read cycle. One might expect that the benchmark test with the latter trace would produce a result closer to the peak value observed by *lmt*: for writes it did but not for reads (see Section 4). The squarish shape to the trace was the result of keeping all tasks busy throughout the fixed time interval of the test, regardless of the amount of data moved. This is called "stonewalling."[4] The use of stonewalling tends to hide the stragglers effect rather than explain or address it.

The new benchmark *mib*[5] (for "MPI I/O Benchmark") can better reveal the causes of the stragglers effect. *Mib* resembles a simplified *IOR*, but generates extra data on the real-time operation within a test. *Mib* notes the exact time of each system call for the length of the test and saves this "system call profile" after the test is over. Post-processing the system call profile provides several insights into the behavior of the stragglers. The tests depicted in Figure 4 were run using *mib*, and Figure 4(c) shows the distribution of the duration of the write system calls for the same test traced in Figure 4(a). Notice that there are many system calls taking as long as 100 seconds. Figure 4(d) shows a corresponding distribution for a test about two weeks later, and a week after the test whose trace is in Figure 4(b) (see Figure 5).

In the BGL environment these timings were extremely precise and coordi-

[4]To our knowledge this term was coined in the iozone (http://www.iozone.org/) documentation. A stonewalling iozone test terminates when the first task completes a target amount of I/O, i.e. "first one to the stone wall." The two meanings of the term "stonewall" are nearly the same, though not identical.

[5]http://sourceforge.net/projects/mibtest

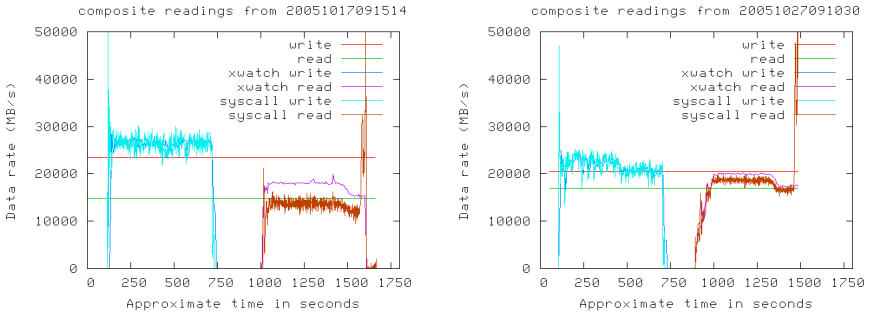
nated, which allowed a detailed map of the progress of the test. In an effort to better understand the stragglers effect the Lustre team tried visualizing the system call profile in a number of ways. Figure 5 shows one useful technique. In those diagrams each task is a separate trace. As each system call is completed the trace moves right by the amount of data written and up by the length of time it takes. The data for figure 5(a) comes from a test run an hour prior to the one in figure 4(a) and (c), and was nearly identical. Figure 5(b) is based on the same data as in figure 4(d). In Figure 5(a) all the the tasks stop when they have transferred the predetermined amount of data, thus they have an even right edge. In Figure 5(b), a stonewalling test, all the the tasks stop after the same amount of time, thus they have an even top edge.

Figure 5 shows two very different system call profiles, both showing stragglers. A close examination of Figure 5(a) shows that some tasks are making very little progress until the bulk of the test is over. Note also the wavy nature of the slower tasks. Finally, there are a small number of tests that progress especially slowly right to the end of the test. It was this diagram, and others like it, that lead the Lustre team to two key insights. First, the tasks were receiving severe back-off penalties from the TCP/IP retry/back-off algorithm. Those back-offs can rise to as much as 120 seconds, which explains why some tasks are very slow to get started. Second, the placement of target files can lead to resource contention. More on the latter shortly. Both phenomena lead to the long tail to the distribution in Figure 4(c). The delayed starts and wavy traces are a direct consequence of the fact that the network switches are oversubscribed and under a coordinated, heavy load. By patching the kernel to make the back-off algorithm tunable, and by making the retries rapid and persistent, this behavior was fixed. The measured data rates improved by about 10%.

The tasks that remained slow from start to finish turned out to be those that inadvertently were assigned to the same server from the same I/O node. The I/O nodes proved to be a serious bottle-neck due to the limited amount of memory in each.

In Figure 5(b) the short, persistent back-off settings were in place. Note that the "waviness" is much smaller. In this diagram some tasks proceed through the test quickly and others a little slower. It turns out that the mapping of which task puts its file on which server can be important. First, if a server gets assigned more files than average it will take longer to serve all requests. Second, if an I/O node is responsible for two tasks both assigned to the same server, as with the really slow tasks in figure 5(a), there is a memory contention issue that will reduce performance rather markedly. An early modification to the Lustre metadata server code produced round-robin assignment of files to servers to avoid overburdening any particular server. This also necessitated a change in test methodology, since all the files needed to be created at one go to let the round-robin assignment work.

The Lustre team began preallocating files and manually verifying they were evenly distributed. Additional careful arrangement insured that no two tasks "behind" the same I/O node were assigned to the same server. The careful arrangement of files and the use of stonewalling made the benchmarking results



(a) Read-ahead set too high

(b) A well behaved test

Figure 6: Data rates as measured by the test results, the *lmt* monitor, and the system calls profile

an artificial measure of the systems overall performance. On the other hand this artificial testing became completely repeatable. That repeatability was itself helpful in tuning and analyzing the I/O subsystem.

4 Composite graphs

The foregoing changes brought the write rate up to an almost acceptable level, but the read data rate still languished. Solving that problem became the team's next priority.

The data in the system call profile for a test may be used to calculate the second-by-second data rate experienced within the MPI tasks. Figure 6(a) combines that measure of instantaneous data rate with the rates supplied by *lmt* and the result of the test. The two flat, horizontal lines represent the write (23 GB/s) and read (15 GB/s) rates reported by the test. *lmt* readings were recorded every 5 seconds and are represented by the lower frequency traces. The higher frequency trace is the second-by-second progress calculated from the system call profile. During the write test the system call data and *lmt* data are almost coincident. The initial spike is the client-side cache filling. In the read test the *lmt* data is consistently above the system call data by about 4 GB/s.

Figure 6(b) shows a similar test about ten days later once the team had begun experimenting with adjustments to the read-ahead settings on the I/O nodes. The data rate measured by *lmt* at the servers is nearly the same, but the data rate seen by the tasks is much higher. Previously, the read-ahead setting was too high and data was being delivered to the I/O nodes only to be discarded unused.

The ability to visualize the behavior of the read-ahead settings allowed the

Figure 7: Primary group exhausted

team to identify the problem in the first place and to tune the settings for optimum effect.

5 Related Work

The *Tau* project[7] is a generalized mechanism for profiling and instrumenting code. The data that it produces may be consumed by visualization tools like *PerfSuite*[6] and *KOJAK*[7]. Similarly, the *plastic file system*[8] is a mechanism for intercepting and logging file system operations. The later can operate transparently to the application. Since these tools are logging information to the I/O subsystem, one must take care that the I/O under observation is not perturbed.

6 Conclusion

The initial under-performance of the file system was, to an extent, a problem of testing methodology. Once the methodological issues were resolved significant progress was made in improving the file system code and in refining the file

[6] http://perfsuite.ncsa.uiuc.edu/
[7] http://www.fz-juelich.de/zam/kojak/
[8] http://plasticfs.sourceforge.net/

system and kernel tunings. Over the course of several months the performance of BGL's file system improved from about 15 GB/s for writes and about 12 GB/s for reads to 29 GB/s for writes and 24 GB/s for reads.

The improved testing methodology that emphasized testing peak performance in a repeatable fashion allowed the Lustre team to identify bottlenecks and establish tunings for optimum performance. The ability to visualize the performance of the I/O subsystem was instrumental in enabling the Lustre team to identify and address performance issues. After numerous incremental improvements, the observed and repeatable results of tests agreed well with expected values.

A careful review of the diagrams included shows that there is more going on. Some features of the graphs are well understood, though not discussed in this paper. One example is the sag in performance after about 400 seconds in Figure 7, which is dramatic in the read trace and also visible to a lesser degree in the write trace. Understanding and addressing that phenomenon, which involves inode and block placement in the servers' underlying EXT3 file systems, is a subject worthy of a paper in its own right. Similarly, the drop in write performance from figure 6(a) to figure 6(b), when things were supposed to be getting better, is also well understood, though again we don't discuss it here. Another topic not covered here is the DDN command queuing, which proved critical in closing the last few percent gap between theoretical and tested performance values. Finally, what's with the dip in the read rate in figure 5? It is not present in later tests, and one may speculate where it came from and where it went.

7 Acknowledgments

The authors would like to thank the members of the Lustre team for their efforts in bringing the BGL Lustre file system up to its final excellent performance. At the time the BGL integration was taking place the Lustre team at LLNL included Brian Behlendorf, Richard Hedges, Terry Heidelberg, Marc Stearman, Andrew Uselton, Herb Wartens, and Ira Weiny. Chris Morrone, a former member of the team, is the author of the *lmt* utility. Todd Inglett and Jeff Parker of IBM explained the inner workings of the BGL I/O nodes. Kim Cupps and Keith Fitzgerald of LLNL were helpful in supporting and encouraging the team's efforts. The LLNL networking team - Jay Drew, Jon Hammond, Bryan Lawver, and Joe Slavec - were enormously helpful in analyzing the federated GigE network. We would also like to thank the friendly people at Cluster File Systems for their diligent efforts at understanding our bugs and improving the Lustre file system.

References

[1] Adiga, N., and et al. An overview of the bluegene/l supercomputer. In

Proceedings of IEEE/ACM Supercomputing (Nov. 2002).

[2] Braby, R. L., Garlick, J. E., and Goldstone, R. J. Achieving order through chaos: the llnl hpc linux cluster experience. Tech. rep., Lawrence Livermore National Lab. http://www.llnl.gov/linux/ucrl-jc-153559.html.

[3] CFS. Lustre: A scalable, high performance file system. Tech. rep., Cluster File Systems. http://www.lustre.org/docs.html.

[4] IBM. An overview of the bluegene/l supercomputer. Tech. rep., IBM. Whitepaper available at http://www-fp.mcs.anl.gov/bglconortium.

[5] Laifer, R. Experiences & performance of sfs/lustre cluster file system in production. Tech. rep., Computing Centre (SSCK) University of Karlsruhe. http://www.rz.uni-karlsruhe.de/rz/docs/Lustre/ssck_sfs_hpccn4_20050510.pdf.

[6] Ross, R., Moriera, J., Cupps, K., and Pfeiffer, W. Parllel i/o on the ibm blue gene/l system. Tech. rep., BGL consortium. http://www-fp.mcs.anl.gov/bgconsortium/filesystems.htm.

[7] Shende, S. S., and Malony, A. D. The tau parallel performance system. *Int. J. High Perform. Comput. Appl. 20*, 2 (2006), 287–311.

[8] Yu, W., Noronha, R., Liang, S., and Panda, D. K. Benefits of high speed interconnects to cluster file systems: A case study with lustre. Tech. rep., Network-Based Computing Lab, Dept. of Computer Sci. & Engineering, Ohio State University. http://ieeexplore.ieee.org/iel5/10917/34366/01639564.pdf.